Semantic robust parsing for noun extraction from natural language queries

Afzal Ballim and Vincenzo Pallotta
MEDIA group: Laboratoire d'Informatique Théorique (LITH)
École Polytechnique Fédérale de Lausanne (EPFL)
IN-F Ecublens 1015 Lausanne (Switzerland)
Phone:+41-21-693 52 97 Fax:+41-21-693 52 78
{ballim,pallotta}@di.epfl.ch

Abstract

This paper describes how robust parsing techniques can be fruitful applied for building a query generation module which is part of a pipelined NLP architecture aimed at process natural language queries in a restricted domain. We want to show that semantic robustness represents a key issue in those NLP systems where it is more likely to have partial and ill-formed utterances due to various factors (e.g. noisy environments, low quality of speech recognition modules, etc...) and where it is necessary to succeed, even if partially, in extracting some meaningful information.

1 Introduction

The domain we are concerned with in our case study is the interaction through speech with information systems. The availability of a large collection of annotated telephone calls for querying the Swiss phone-book database (i.e the Swiss French PolyPhone corpus [8]) allowed us to experiment our recent findings in robust text analysis obtained in the context of the Swiss National Fund research project ROTA (Robust Text Analysis), and in the Swisscom funded project ISIS (Interaction through Speech with Information Systems). Within this domain, the goal is to build a valid query to an information system, using limited world knowledge of the domain in question. Although a task like this may, at its simplest, be performed quite effectively using heuristic methods such as keyword spotting, such an approach is brittle, and does not scale up easily in the case of conducting a dialogue.

1.1 Problem specification

In this section we will give an informal specification for the problem of processing telephone calls for querying a phone-book database.

1.1.1 Swiss French PolyPhone Database

These database contains 4293 simulated recordings related to the "111" Swisscom service calls (e.g. "rubrique 38" of the calling sheet [8]). Each recording consists of 2 files, one ASCII

text file corresponding to the initial prompt and the information request and one data file containing the sampled sound version. As far as the address fields are concerned, the data in the PolyPhone database are unfortunately not tagged and even not consistent. Prompts and information requests expressed by users have been extracted from the files an regrouped into a single representation in the following format:

```
id:cd1/b00/f0000o06:sid17733
prompt:1
adr1:MOTTAZ MONIQUE
adr2:rue du PRINTEMPS 4
adr3:SAIGNELEGIER
text[123]: Bonjour j'aimerais un numéro de téléphone à Saignelegier c'est Mottaz m o deux ta z Mo
sample:0.200000:10.820000:88160:42801
```

where currently, the corresponding lines in text file are processed with the following heuristic:

id identifies the original location of the file in the CD-ROM.

prompt identifies both the type of prompt asking the user for posing the query (e.g. n. 1 corresponds to "*Veuillez maintenant faire comme si vous étiez en ligne avec le 111 pur demander le numéro de téléphone de la personne imaginaire dont les coordonnées se trouvent ci-dessous:*").

adr1 corresponds to the *name*

adr2 corresponds to the *address* if line 3 is not empty and *town* otherwise

adr3 corresponds to the *town* if not empty.

text corresponds to the text transcription. The number in square brackets is the total number of chars in the request.

sample groups the information for the sampled sound version of the request.

This heuristic seems to perform quite well but a more thorough and exhaustive evaluation still needs to be carried out. The main problem remains in finding enough information about the *original* data in order to be able to perform the validation automatically.

1.1.2 The frame schema

Concerning the structure in the Swiss Phone-book database, we assumed it is the same as the one that appears on the web (e.g. http://www.ife.ee.ethz.ch/cgi-bin/etvq/f), namely (one field per line):

```
Nom de famille / Firme
Prénom / Autres informations
No de téléphone
Rue, numéro
NPA, localité
```

We chose to provide further information which are not available at web level but which can be used to form the query. The full frame description is given below[1]:

[Caller]

- **Title:**
- **Name:**
- **Locality:**

Target_Identification

- **Name** (default: Person)
  - ***Person**
    - **Family name:**
    - **[Title]:**
    - **[First name]:**
    - **[Second name]:**
    - **[Occupation]**
      - **Description:**
      - **[Class]:** [*yellow pages categories*]
  - ***Company**
    - **Name:**
    - **[Description]:**
    - **[Category]:** [*yellow pages categories*]
    - **[Owner]:**
    - **[Contact person]:** [repres., direction, secretariat, ...]

Target_Address

- **[Appart n.]:**
- **[Street n.]:**
- **[Building]:**
- **[Street name]:**
- **[Village]:**
- **[NPA]:**
- **Loc_type:**
- **Locality** (at least one of the sub-fields)
  - **City:**
  - **"Environs":**
  - **Region:**
  - **Canton:**
  - **Telephone prefix:**

Request type

- **Phone type:** (default: standard) [standard, privé, fax, natel]
- **Request status:** (default: ok) [ok, ill-formed, missing-information, ...]

---

[1] Bracketted slots are optional.

One point still remains unclear about the PolyPhone database (as no answers where found in [3, 8]): what was the set of annotation used for the transcription of utterances? Several speech annotations such as "<hesitation>" appear in the text. Was it systematic? Are there other such markers? It is possible to rely on prosodic informations? In the first phase of the project we simply skipped these informations but we guess that they could be of great help in disambiguating interpretations of strict adjacent sequences of names such as in utterances like "*j'amerais le numéro de téléphone de Vedo-Moser Brigitte Brignon Baar-Nendaz*".

1.2 Query analysis

The processing of the corpus data is performed at various linguistic levels performed by modules organized into a pipeline. Each module assumes as input the output of the preceding module. The main goal of this architecture is to understand how far it is possible go without using any kind of feedback and interactions among different linguistic modules.

1.2.1 Morpho-Syntactic analysis

At a first stage, morphological and syntactic processing is applied to the output from the *speech recognizer* module which usually produces a huge word-graph hypothesis. Low-level processing (morphological analysis and tagging) were performed by ISSCO (Institute Dalle Molle, University of Geneva) using tools that were developed in the European Linguistics Engineering project MULTEXT. For syntactic analysis, ISSCO developed a Feature Unification Grammar based on the ELU formalism [9] (i.e. an extension of PATRII grammars) and induced by a small sample of the Polyphone data. This grammar was taken by another of our partners (the Laboratory for Artificial Intelligence of the Swiss Federal Institute of Technology, Lausanne) and converted into a probabilistic context-free grammar, which was then initially trained with a sample of 500 entries from the Polyphone data. The forest of syntactic trees produced by this phase will be used to achieve two goals:

1. The n-best analyses are use to disambiguate speech recognizer hypotheses
2. They served as the input for the robust semantic analysis that we performed, that had as goal the production of query frames for the information system.

1.2.2 Semantic annotations

While the semantic analysis will in general reduce the degree of ambiguity found after syntactic analysis, there remains the possibility that it might *increase* some degree of ambiguity due to the presence of coherent senses of words with the same syntactic category (e.g., the word "Geneva" can refer to either the canton or the city).

1.2.3 Semantic robust analysis and frame filling

The component that deals with such input is generally referred to as a *robust analyzer*. Although robustness can be considered as being applied at either a syntactic or semantic level, we believe it is generally at the semantic level that it is most effective. This robust analysis needs a model of the domain in which the system operates, and a way of linking this model

to the lexicon used by the other components. It specifies semantic constraints that apply in the world and which allow us to rule out incoherent requests (for instance). The degree of detail required of the domain model used by the robust analyzer depends upon the ultimate task that must be performed — in our case, furnishing a query to an information system. Taking the assumption that the information system being queried is relatively close in form to a relational database, the goal of the interpretative process is to furnish a query to the information system that can be viewed in the form of a frame with certain fields completed, the function of the querying engine being to fill in the empty fields.

One way in which the interface could interact with the querying system would be to submit such a frame at the end of the analysis process without performing any coherency checking. The advantage of this method is that the model of the domain of queries that is required by the interface can be limited. However, such an approach has two major disadvantages:

- the result of incorrectly formulated queries may be completely uninterpretable or erroneous, and the interface system would have no basis for evaluating the quality of such replies, or how to aid the user in formulating a better one;
- there might be a number of possible frames that could be submitted for any instance of a user utterance/query, and this number might be reducible by application of a model of coherent queries.

We will, therefore, presume that queries must be classified by the interface into three categories:

1. the query is correct — the fields of the frame which must be completed contain semantically valid data. The query may be submitted;
2. incomplete queries — certain necessary fields cannot be unambiguously filled in, and so a system-initiative dialogue can be invoked to furbish the necessary information to create a correct query;
3. incoherent queries — information in the fields of the frame is not coherent with the interfaces model of the domain. An error dialogue must be invoked.

The last query category is the most complex, since it requires a domain model sufficiently rich to decide whether a query is outside of the domain, or inside the domain but violating certain semantic constraints. In addition, it requires relatively complex dialogue management as the corrective dialogue may involve resolution of miscomprehension by either the system or the user.

2 Computational logic for robust analysis

What has been considered to be an advantage using *logic-based* programming languages is the symbol processing capability and the way of abstracting from the actual implementation of needed data structures. *Definite Clause Grammars* come to mind when relating Logic Programming and Natural Language Processing. This is of course one of the best couplings

between Computational Linguistics and Logic to support both (i) the development of linguistic models of Natural Language (Computational Linguistics) and (ii) the design of real life applications (Language Engineering).

The main drawback to this approach is efficiency, but it is not the only one. In recent years several efforts have been done to improve efficiency of logic and functional programming languages by means of powerful abstract machines and optimized compilers. Sometimes, efficiency recovery leads to introduction of non-logical features in the language and the programmer should be aware of it in order to exploit it in the development of his or her applications (i.e. cut in logic programming).

An important question to ask is: "how can computational logic contribute to robust discourse analysis ?". A partial answer to this question is that currently logic-based programming languages are able to integrate in an unifying framework all or most of the techniques necessary for robust text analysis. Furthermore this can be done in a rigorous "mathematical" fashion. In this sense robustness is related to correctness and provability with respect to the specifications. A NLP system developed within a logical framework has a predictable behavior which is useful in order to check the validity of the underlying theories.

2.1 Left-corner Head-driven Island Parser

LHIP [5, 14] is a system which performs robust analysis of its input, using a grammar defined in an extended form of the Definite Clause Grammar formalism used for implementation of parsers in Prolog. The chief modifications to the standard Prolog 'grammar rule' format are of two types: one or more right-hand side (RHS) items may be marked as 'heads', and one or more RHS items may be marked as 'ignorable'.

LHIP employs a different control strategy from that used by Prolog DCGs, in order to allow it to cope with ungrammatical or unforeseen input. The behavior of LHIP can best be understood in terms of the complementary notions of **span** and **cover**. A grammar rule is said to produce an island which **spans** input terminals t_i to t_{i+n} if the island starts at the i^{th} terminal, and the $i+n^{th}$ terminal is the terminal immediately to the right of the last terminal of the island. A rule is said to **cover** m items if m terminals are consumed in the span of the rule. Thus $m \leq n$. If $m = n$ then the rule has completely covered the span.

As implied here, rules need not cover all of the input in order to succeed. More specifically, the constraints applied in creating islands are such that islands do not have to be adjacent, but may be separated by non-covered input. There are two notions of non-coverage of the input: **unsanctioned** and **sanctioned** non-coverage. The former case arises when the grammar simply does not account for some terminal. Sanctioned non-coverage means that special rules, called "*ignore*" rules, have been applied so that by ignoring parts of the input the islands are adjacent. Those parts of the input that have been *ignored* are considered to have been consumed. These *ignore* rules can be invoked individually or as a class. It is this latter capability which distinguishes *ignore* rules from regular rules, as they are functionally equivalent otherwise, but mainly serve as a notational aid for the grammar writer.

Strict adjacency between RHS clauses can be specified in the grammar. It is possible to define global and local thresholds for the proportion of the spanned input that must be covered by rules; in this way, the user of an LHIP grammar can exercise quite fine control over the required accuracy and completeness of the analysis.

A chart is kept of successes and failures of rules, both to improve efficiency and provide a means of identifying unattached constituents. In addition, feedback is given to the grammar writer on the degree to which the grammar is able to cope with the given input; in a context of grammar development, this may serve as notification of areas to which the coverage of the grammar might next be extended. Extensions of Prolog DCG grammars in LHIP permit:

1. nominating certain RHS clauses as heads;
2. marking some RHS clauses as being optional;
3. invocation of *ignore* rules;
4. imposing adjacency constraints between two RHS clauses;
5. setting a local threshold level in a rule for the fraction of spanned input that must be covered.

A threshold defines the minimum fraction of terminals covered by the rule in relation to the terminals spanned by the rule in order for the rule to succeed. For instance, if a rule spans terminals t_i to t_{i+n} covering j terminals in that span, then the rule can only succeed if $j/n \geq T$.

The following is an example of a LHIP rule. At first sight this rule appears left recursive. However, the sub-rule "`conjunction(Conj)`" is marked as a head and therefore is evaluated before either of "`s(Sl)`" or "`s(Sr)`". Presuming that the conjunction-rule does not end up invoking (directly or indirectly) the s-rule, then the s-rule is not left-recursive.

```
s(conjunct(Conj,Sl,Sr)) ~~>
          s(Sl)
         *conjunction(Conj),
          s(Sr).
```

LHIP provides a number of ways of applying a grammar to input. The simplest allows one to enumerate the possible analyses of the input with the grammar. The order in which the results are produced will reflect the lexical ordering of the rules as they are converted by LHIP. With the threshold level set to 0, all analyses possible with the grammar by deletion of input terminals can be generated. By setting the threshold to 1, only those partial analyses that have no unaccounted for terminals within their spans can succeed. Thus, supposing a suitable grammar, for the sentence *John saw Mary and Mark saw them* there would be analyses corresponding to the sentence itself, as well as *John saw Mary*, *John saw Mark*, *John saw them*, *Mary saw them*, *Mary and Mark saw them,* etc. By setting the threshold to 1, only those partial analyses that have no unaccounted for terminals within their spans can succeed. Hence, *Mark saw them* would receive a valid analysis, as would *Mary and Mark saw them*, provided that the grammar contains a rule for conjoined NPs; *John saw them*, on the other hand, would not. As this example illustrates, a partial analysis of this kind may not in fact correspond to a true sub-parse of the input (since *Mary and Mark* was not a conjoined subject in the original). Some care must therefore be taken in interpreting results.

This rule illustrates a number of features: *negation*, and *optional forms*. The rule will only succeed if (with respect to the area of input in which it might occur) there is a noun with no determiner. In addition, there can be optional adjectives before the noun.

```
np(propernoun(N,Mods)) ~~>
                ~ determiner(_),
                 (? adjectives(Mods) ?),
                * noun(N).
```

This rule illustrates the use of disjunction and embedded Prolog code. It should be noted that within the scope of a disjunction or negation, a head is local to the disjunct or negation.

```
noun(X) ~~>
     ( * @pussy, (? @cat ?); * @cat),
     {X=cat}.
```

This rule illustrates a typical use of adjacency, to specify compound nouns. Adjacency is not restricted such a use however, but may generally be used anywhere.

```
noun(missionary_camp) ~~> @missionary : @camp.
```

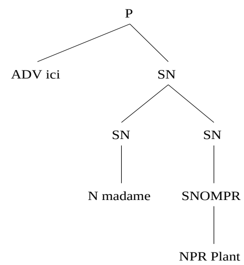
A number of tools are provided for producing analyses of input by the grammar with certain constraints. For example, to find the set of analyses that provide maximal coverage over the input, to find the subset of the maximal coverage set that have minimum spans, and to find the find analyses that have maximal thresholds. In addition, other tools can be used to search the chart for constituents that have been found but are not attached to any complete analysis. The conversion of the grammar into Prolog code means that the user of the system can easily develop analysis tools that apply different constraints, using the given tools as building blocks.

3 Implementation of the semantic module

In our approach we try to integrate the above principles in our system in order to effectively compute hypotheses for the frame filling task. This can be done by building a lattice of *frame filling hypotheses* and possibly selecting the best one. Hypotheses are typically sequences of proper names. The lattice of hypotheses is generated by means of LHIP *discourse grammar*. This type of grammar is used to extract *names chunks* and assemble them into the hypothesized frame structure.

3.1 Tree-paths representation

Parse trees obtained from the previous module are encoded into a path representation which allows us to easily specify constraints over the tree structure. A *path-sentence* is a list of *path-words* which in turn are compound terms of the type `terminal(word, path)` where *word* is a constant term and *path* is a list of arc identifiers that is compound terms `'cat'(#number_of_nodes, #node, #identifier)` uniquely identifying an arc in the parse tree. The functor `'cat'` is a category name and its arguments are integer positive numbers. For instance the representation of the parse tree:



is given by:

```
[terminal(ici,['ADV'(1,1,14),'P'(2,1,12),'P'(2,1,11)),
terminal(madame,['N'(1,1,19),'SN'(1,1,17),'SN'(2,1,16),'P'(2,2,15),'P'(2,1,11)),
terminal('Plant',['NPR'(1,1,24),'SNOMPR'(1,1,22),'SN'(1,1,21),'SN'(2,2,20),'P'(2,2,15),'P'(2,1,11)].
```

Using this representation it is possible to define a grouping operator (e.g. `group/2`) which given a sequence of adjacent names finds the subsequence of words having the least common ancestor which is closer than the least common ancestor (e.g. `lca/2`) of the given sequence. These two operators are very useful for imposing structural knowledge constraints and they are straightforwardly defined as PROLOG programs by:

```
lca([terminal(_,W)],W).
lca([terminal(_,W)|R],P) :-
        lca(R,P1),
        prefix_path(P1,P),
        prefix_path(W,P),!.

group([],[]).
group(L,X) :-
        lca(L,P),
        proper_sublist(L,X), length(X,N), N>1,
        lca(X,P1),
        proper_sublist(P1,P).

prefix_path(A,A).
prefix_path([_|B],C) :-
        prefix_path(B,C).
```

3.2 Discourse markers

Discourse segments allow us to model dialog by a set of pragmatic concepts (dialogue acts) representing what the user is expected to utter (for example initiation of a dialogue: *init*, expression of gratitude: *thank*, and demand for information: *request*, etc.) and in that way are useful for reducing the syntactic and semantic ambiguity. These are domain-dependent and must be defined for a given corpus. For their definition, we intend to follow the experiments done in the context of Verbmobil (see for example [11, 12]). In our specific case identifying special words serving both as separators among logical subparts of the same sentence and as introducers of semantic constituents allows us to search for name sequences to fill a particular slot only in interesting part of the sentence. One of the most important separator is the *announcement-query separator*. The LHIP clauses defining this separator can be one or more word covering rule like for instance:

```
ann_query_separator #1.0 ~~>
        @terminal('téléphone',_).
ann_query_separator #1.0 ~~>
        ( @terminal('numéro',_):
          @terminal('de',_):
          (? @terminal('téléphone',_) ?)).
```

As an example of semantic constituents introducers we propose here the

```
street_intro([T,Prep,Det],1) #1.0 ~~>
        * street_type(T),
        preposition(Prep),
        determiner(Det).
```

which make use of some word knowledge about street types coming from an external thesaurus like:

```
street_type(terminal(X,P)) ~~>
        @terminal(X,P),
        {thesaurus(street,W),member(X,W)}.
```

3.3 Generation of hypotheses

The generation of hypotheses for filling the frame is performed by: composing weighted rules, assembling chunks and filtering possible hypotheses.

3.3.1 Weighted rules

The main assumption on which probabilistic approach to NLP is based, is that language is considered as being a random phenomenon with its own probability distribution function: *coverage* is often translated as *expectation* in a probabilistic sense. Changing perspective and considering language just as an *uncertain* and *imprecise* phenomenon and understanding as a *perception* process, it is naturally to think of *fuzzy* models of language (see [13] and [4]). Recently, fuzzy reasoning has been partially integrated into a CLP paradigm (see [15]) in order to deal with so called *soft constraints* in weighted *constraint logic grammars*. We tried to get some inspiration from the above proposal for integrating fuzzy logic and parsing to compute weights to assign to each frame filling hypotheses. Each LHIP rule returns a confidence factor together with the sequence of names. The confidence factor for a rule can be either assigned statically (e.g. to pre-terminal rules) or they can be computed composing recursively the confidence factors of sub-constituents. Confidence factors are combined choosing the minimum among confidences of each sub-constituents. It is possible that there is no enough information for filling a slot. In this case the grammar should provide a mean to provide an empty constituent when all possible hypothesis rules have failed. This is possible using negation and epsilon-rules in LHIP as showed in the following rules for dealing with street names.

```
found_street_name(L,Conf) #1.0 ~~>
        * street_intro(Intro,Conf),
        name_list(X),
        {append(Intro,X,L)}.
found_street_name(X,0.3) ~~>
        * name_list(X).
hyp_street_name(Street,Conf) ~~>
        * found_street_name(Street,Conf).
hyp_street_name([],1) ~~>
        ~found_street_name(_,_),
        lhip_true.
```

where `name_list(X)` accounts for sequence of adjacent proper names and `lhip_true` corresponds to the empty sequence.

Observe that in this particular case there is no need to select the minimum confidence factor from the sub-constituents of the rule `found_street_name` since we have only `street_intro(Intro,Conf)` which propagates its confidence factor.

3.3.2 Chunk assembling

The highest level constituent is represented by the whole frame structure which simply specifies the possible orders of chunks relative to slot hypotheses. A rule for a possible frame hypothesis is:

```
frame(Caller_title, Caller_name,
      Target_title, Target_name,
      Street_name, Street_number,
      Locality, Weight)
             ~~> hyp_caller(Caller_title,Caller_name,C1),
                 * ann_query_separator,
                 hyp_target(Target_title,Target_name,C2),
                 * location_intro,
                 hyp_street_name(Street_name,C3),
                 hyp_street_number(Street_number,C4));
                 hyp_locality_name(Locality,C5),

                 {minlist([C1,C2,C3,C4,C5],Weight)}.
```

In this rule we specify a possible order of chunks interleaved by separators and introducers. The computation of global weight may be more complex than the above rule which uses simply the minimum of each hypothesis confidence values. In this case we did not provide any structural constraint (e.g. preferring names chunks belonging to the minimal common sub-tree or those having the longest sequence of name belonging to the same sub-tree).

3.3.3 Filtering and query generation

The obtained frame hypotheses can be further filtered by both using structural knowledge (e.g. constraints over the tree-path representation) and word knowledge. In order to combine the

information extracted from the previous analysis step into the final query representation which can be directly mapped into the database query language we will make use of a frame structure in which slots represent information units or attributes in the database. A simple notion of context can be useful to fill by default those slots for which we have no explicit information. For doing this type of *hierarchical reasoning* we exploit the meta-programming capabilities of logic programming and we used a meta-interpreter which allows multiple inheritance among logical theories [7]. More precisely we made use of the special *retraction* operator "$\prec$" for composing logic programs which allows us to easily model the concept of inheritance in hierarchical reasoning. The expression $P \prec Q$, where P and Q are meta-variables used to denote arbitrary logic programs, means that the resulting logic programs contains all the definition of P except those that are also defined in Q.

The definition of the *isa* operator is obtained combining the retraction operator with the union operator (e.g. $\cup$) that simply make the physical union of two logic programs, by

$$P \; isa \; Q = P \cup (Q \prec P).$$

As an example for the above definition we provide some default definitions which have been used to represent part of the world knowledge in our domain. The *rules* theory contains rules for inferring the locality or the locality type when they are not explicitly mentioned in the query.

rules:

```
locality(City) :-
        caller_prefix(X),
        prefix(X,City).

loc_type(Type) :-
        locality(City),
        gis(City,Type).
```

where `prefix/2` and `gis/2` are world knowledge bases (i.e. a collection of facts grouped in a theory called *kb*) and `caller_prefix/1` can be easily provided from the answer system.

If some information is missing then the system tries to provide some default additional information to complete the query. The following theory contains definition for some mandatory slots which need to be filled in case of incomplete queries, like for instance in the theory *query_defaults*:

query_defaults:

```
identification(person).
phone_type(standard).
loc_type(city).
```

Finally starting from an incomplete query which does not account for the required information we can use deduction to generate the query completion like for instance asking for:

```
?- demo((query isa query_default) ∪ rules ∪ kb), loc_type(X)).
```

4 Conclusions

From a very superficial observation of the human language understanding process, it appears clear that no deep competence of the underlying structure of the spoken language is required in order to be able to process acceptably distorted utterances. On the other hand, the more experienced is the speaker, the more probable is a successful understanding of that distorted input. How can this kind of fault-tolerant behavior be reproduced in an artificial system by means of computational techniques? Several answers have been proposed to this question and many systems implemented so far, but no one of them is capable of dealing with robustness as a whole.

As examples of robust approaches applied to dialogue systems we cite here two systems which are based on similar principles.

In the DIALOGOS human-machine telephone system (see [1]) the robust behavior of the *dialogue management* module is based both on a contextual knowledge base of pragmatic-based expectations and the dialogue history. The system identifies discrepancies between expectations and the actual user behavior and in that case it tries to rebuild the dialogue consistency. Since both the domain of discourse and the user's goals (e.g. railway timetable inquiry) are clear, it is assumed the systems and the users cooperate in achieving reciprocal understanding. Under this underlying assumption the system pro-actively asks for the query parameters and it is able to account for those spontaneously proposed by the user.

In the SYSLID project (see [6]) where a robust parser constitutes the *linguistic component* (LC) of the *query-answering dialogue system* . An utterance is analyzed while at the same time its semantical representation is constructed. This semantical representation is further analyzed by the *dialogue control module* (DC) which then builds the database query. Starting from a *word graph* generated by the speech recognizer module, the robust parser will produce a search path into the word graph. If no complete path can be found, the robust component of the parser, which is an island based chart parser (see [10]), will select the maximal consistent partial results. In this case the parsing process is also guided by a *lexical semantic knowledge base* component that helps the parse in solving structural ambiguities.

We can conclude that robustness in dialogue is crucial when the artificial system takes part in the interaction since inability or low performance in processing utterances will cause unacceptable degradation of the overall system. As pointed out in [2] it is better to have a dialogue system that tries to guess a specific interpretation in case of ambiguity rather than ask the user for a clarification. If this first commitment results later to be a mistake a robust behavior will be able to interpret subsequent corrections as repair procedures to be issued in order to get the intended interpretation.

References

[1] Dario Albesano, Paolo Baggia, Morena Danieli, Roberto Gemello, Elisabetta Gerbino, and Claudio Rullent. Dialogos: a robust system for human-machine spoken dialogue on the telephone. In *Proc. of ICASSP*, Munich, Germany, 1997.

[2] J.F. Allen, B. Miller, E. Ringger, and T. Sikorski. A robust system for natural spoken dialogue. In *Proc. 34th Meeting of the Assoc. for Computational Linguistics*. Association of Computational Linguistics, June 1996.

[3] J.M. Andersen, G. Caloz, and H. Bourlard. Swisscom "advanced vocal interfaces services" project. Technical Report COM-97-06, IDIAP, Martigny, December 1997.

[4] Peter. R.J. Asveld. Towards robustness in parsing - fuzzifying context-free language recognition. In J. Dassow, G. Rozemberg, and A. Salomaa, editors, *Developments in Language Theory II - At the Crossroad of Mathematics*, Computer Science and Biology, pages 443–453. World Scientific, Singapore, 1996.

[5] A. Ballim and G. Russell. LHIP: Extended DCGs for Configurable Robust Parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 501 – 507, Kyoto, Japan, 1994. ACL.

[6] Manuela Boros, Gerhard Hanrieder, and Ulla Ackermann. Linguistic processing for spoken dialogue systems - experiences made in the syslid project -. In *Proceedings of the third CRIM-FORWISS Workshop*, Montreal, Canada, 1996.

[7] A. Brogi and F. Turini. Meta-logic for program composition: Semantic issues. In K.R. Apt and F.Turini, editors, *Meta-Logics and Logic Programming*. The MIT Press, 1995.

[8] G. Chollet, J.-L. Chochard, A. Constantinescu, C. Jaboulet, and Ph. Langlais. Swiss french polyphone and polyvar: Telephone speech database to model inter- and intra-speaker variability. Technical Report RR-96-01, IDIAP, Martigny, April 1996.

[9] Dominique Estival. *ELU User Manual*. ISSCO, Geneva – Switzerland, 1990. http://issco-www.unige.ch/publications/working-papers/.

[10] G. Hanrieder and G Görz. Robust parsing of spoken dialogue using contextual knowledge and recognition probabilities. In *Proceedings of the ESCA Tutorial and Research Workshop on Spoken Dialogue Systems – Theories and Applications*, pages 57–60, Denmark, May 1995.

[11] S. Jekat, A. Klein, E. Maier, I. Maleck, M. Mast, and J.J. Quantz. Dialogue acts in vermobil. Verbmobil Report 65, DFKI, 1995.

[12] R. Kompe, A. Kiebling, T. Kuhn, M. Mast, H. Niemann, E. Nöth, K. Ott, and A. Batliner. Prosody takes over: A prosodically guided dialog system. Verbmobil report 47, DFKI, 1994.

[13] E.T. Lee and L.A. Zadeh. Note on fuzzy languages. *Information Science*, 1:421–434, 1969.

[14] C. Lieske and A. Ballim. Rethinking natural language processing with prolog. In *Proceedings of Practical Applications of Prolog and Practical Applications of Constraint Technology (PAPPACTS98)*, London,UK, 1998. Practical Application Company.

[15] Stefan Riezler. Quantitative constraint logic programming for weighted grammar applications. In *Logical Aspects of Computational Linguistics (LACL'96)*, LNCS. Springer, 1996.